Chương Trình Con và Hàm

(Sub-Program and Function)

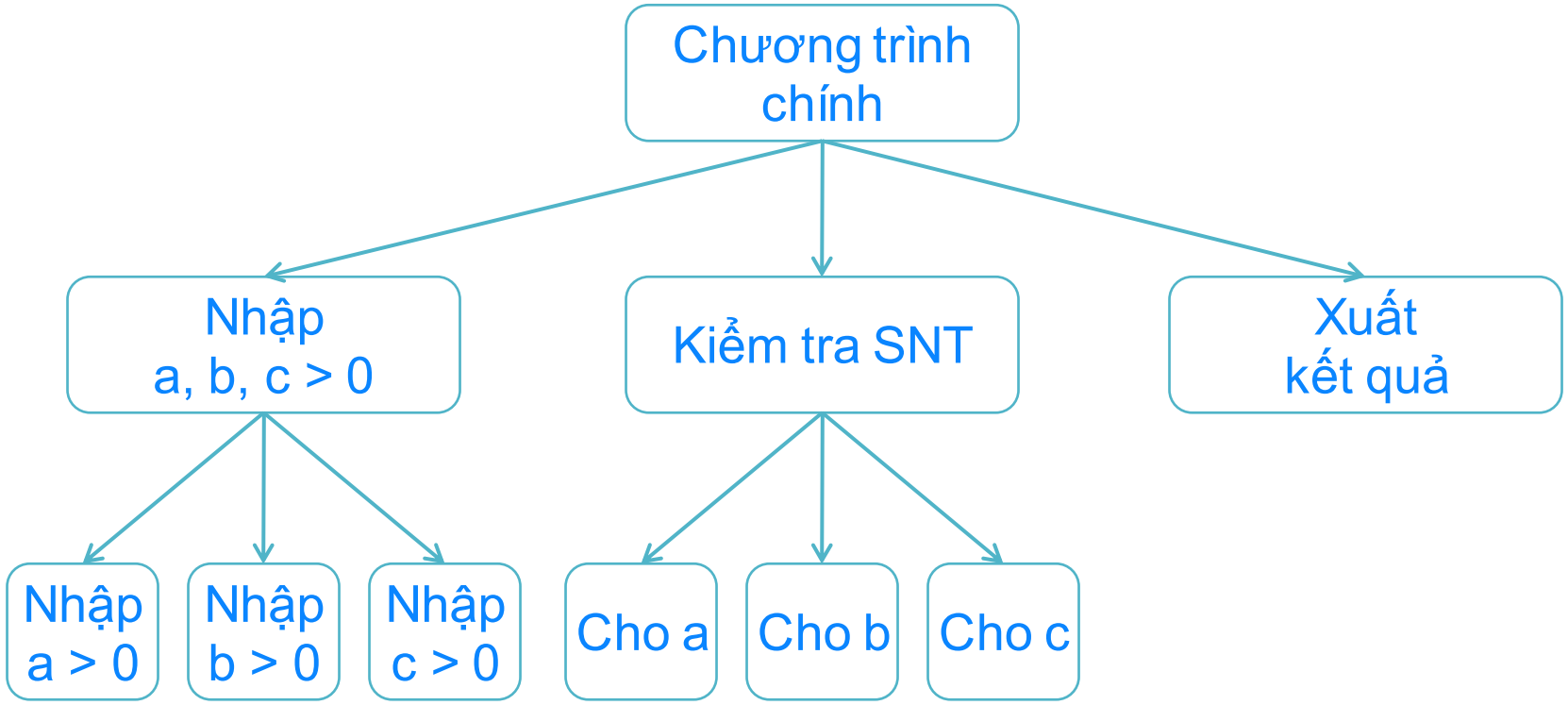
Trịnh Tấn Đạt
Khoa CNTT - Đại Học Sài Gòn
Email: trinhtandat@sgu.edu.vn
Website: https://sites.google.com/site/ttdat88/

Nội dung

- Chương trình con
- Khai báo và định nghĩa hàm; nguyên mẫu hàm
- Gọi hàm
- Truyền tham số cho hàm
- Hàm có giá trị trả về và hàm void
- Phạm vi của biến
- Marco vs. Hàm
- Các ví dụ minh họa
- Bài Tập

Đặt vấn đề

- Viết chương trình nhập vào 03 số a,b,c. Kiểm tra các số nào là số nguyên tố?



Đặt vấn đề

- Khi đó:
  - Cần phải viết 3 đoạn code cho việc nhập a,b,c với điều kiện là số nguyên dương

```
do {
      cout<<"Nhap mot so nguyen duong:";
       cin>>a;
    } while (a <= 0);
```

```
do {
      cout<<"Nhap mot so nguyen duong:";
       cin>>b;
    } while (b <= 0);
```

```
do {
      cout<<"Nhap mot so nguyen duong:";
       cin>>c;
    } while (c <= 0);
```

Đặt vấn đề

- Khi đó:
  - Cần phải viết đoạn code cho việc kiểm tra a là số nguyên tố?
  - Cần phải viết đoạn code cho việc kiểm tra b là số nguyên tố?
  - Cần phải viết đoạn code cho việc kiểm tra c là số nguyên tố?

```
if(a<2)
        cout<<"a la khong la SNT<<endl;
else
{
        for(int i=2; i<=sqrt(a);i++)
        {
             if(a%i ==0)
             {  cout<<"a la khong la SNT<<endl;
                  break;
             }
        }
         cout<<"a khong la SNT<<endl;
}
```

Đặt vấn đề

❖ Giải pháp => Viết 1 lần và sử dụng nhiều lần

- Đoạn lệnh nhập tổng quát, với n
- Đoạn lệnh kiểm tra số nguyên tố tổng quát, với n
- Thay thế các giá trị a, b, c vào n cho từng trường hợp

⇒ Phần code chung có khả năng tái sử dụng

Chương trình con

- **Chương trình con (sub-program):** là một phần mã trong một chương trình lớn hơn, phần mã này thực hiện một tác vụ cụ thể và tương đối độc lập với phần mã còn lại.
- Một chương trình con thường được viết mã sao cho nó có thể được **gọi nhiều lần từ nhiều nơi** trong thời gian chạy của một chương trình (có thể được gọi bởi chính nó).
- Các chương trình con thường được **tập trung thành các thư viện**, là một cơ chế quan trọng cho việc chia sẻ và tái sử dụng mã.

Chương trình con

- Chương trình con có 2 loại: Thủ tục (**Procedure**) và hàm (**Function**):
  - **Thủ tục (PROCEDURE):** Dùng để thực hiện một hay nhiều nhiệm vụ nào đó (thường không có giá trị trả về).
  - **Hàm (FUNCTION):** Trả về một giá trị nào đó (có kiểu vô hướng, kiểu string hoặc kiểu con trỏ). Hàm có thể sử dụng trong các biểu thức.

Chương trình con

- Lưu ý: Khi nào thì nên dùng thủ tục/hàm:
  - Dùng hàm:
    - Kết quả của bài toán **trả về 1 giá trị duy nhất** (kiểu vô hướng, kiểu string hoặc kiểu con trỏ).
    - Phát biểu gọi CHƯƠNG TRÌNH CON cần **nằm trong các biểu thức tính toán.**
  - Dùng thủ tục:
    - Kết quả của bài toán **không trả về giá trị nào hoặc trả về nhiều giá trị hoặc trả về kiểu dữ liệu có cấu trúc** (Array, Record, File).
    - Phát biểu gọi CHƯƠNG TRÌNH CON **KHÔNG nằm trong các biểu thức tính toán**.

Chương trình con

- Chương trình con được dùng khi xây dựng các chương trình lớn nhằm:
  - giảm đáng kể kích thước và chi phí của một chương trình
  - làm cho chương trình dễ theo dõi,
  - dễ sửa lỗi,
  - nâng cao độ tin cậy của chương trình.
- Một đặc điểm nổi bật của chương trình con là nó có tính đệ quy nhờ thế mà nhiều bài toán được giải quyết dễ dàng.
- Chương trình con trong ngôn ngữ C/C++ là *hàm (function).*

Hàm (function)

- Hàm là một chương trình con bao gồm một khối các câu lệnh thực hiện một nhiệm vụ nhất định, và có thể được gọi khi cần
- Mỗi hàm có một tên (**các hàm trong C không được trùng tên nhau**), một số tham số, và một giá trị trả về.
  - Thư viện của C/C++ hỗ trợ nhiều hàm khác nhau. Ví dụ: cin, printf, pow, sqrt ,..
  - Ngoài ra, hàm có thể do người dùng định nghĩa (đặt tên phải khác tên các hàm trong thư viện C/C++)

Định nghĩa hàm

- Định nghĩa hàm gồm **kiểu trả về, tên hàm, các tham số và thân hàm** (chứa các phát biểu chương trình), thực thi một việc cụ thể.
- Cú pháp: (Dạng định nghĩa hàm)

```
<kiểu trả về> <tên hàm>([danh sách tham số])
{
    <các câu lệnh>;
    [return <giá trị>;]
}
```

trong đó:
– Kiểu trả về (**return_type, còn gọi là *kiểu hàm) tương ứng*** với kiểu của giá trị mà hàm trả về thông qua phát biểu **return.** Kiểu bất kỳ của C (**char, int, long, float,…).** Nếu không trả về thì là **void.**
– Tên hàm (func_name) được đặt theo nguyên tắc đặt tên, nhưng nên đặt tên sao cho dễ hiểu.
– **Danh sách tham số (ParameterList):** mỗi tham số được xác định bởi ***kiểu dữ liệu và tên***. *Các tham số* ***phân cách*** *nhau bởi* ***dấu phẩy***. ***Có thể là danh sách rỗng.***
– Phần thân hàm nằm giữa cặp ngoặc **{ và }.**

Không có dấu chấm phẩy ';' ở cuối khi định nghĩa hàm

Định nghĩa hàm

❖ Lưu ý:

- Nếu không xác định return_type, mặc định sẽ là kiểu int.
- Nếu hàm không trả về giá trị, dùng void (thay cho return_type).

  Ví dụ: int luythua2(int n) ; void display() ;
- Các hàm được định nghĩa không phải theo một thứ tự nào.

Lưu ý: Các bước viết hàm

❖ Cần xác định các thông tin sau đây:

- Tên hàm.
- Hàm sẽ thực hiện công việc gì.
- Các đầu vào (nếu có) - Input
- Đầu ra (nếu có) - Output

Ví dụ:

Ví dụ 1

- Tên hàm: XuatTong
- Công việc: tính và xuất tổng 2 số nguyên
- Đầu vào: hai số nguyên x và y
- Đầu ra: không có

```
void XuatTong(int x, int y)
{
    int s;
    s = x + y;
    cout<<"tong la:"<<s;
}
```

Ví dụ:

Ví dụ 2

- Tên hàm: TinhTong
- Công việc: tính và trả về tổng 2 số nguyên
- Đầu vào: hai số nguyên x và y
- Đầu ra: một số nguyên có giá trị x + y

```
int TinhTong(int x, int y)
{
    int s;
    s = x + y;
    return s;
}
```

Ví dụ:

Ví dụ 3

- Tên hàm: NhapXuatTong
- Công việc: nhập và xuất tổng 2 số nguyên
- Đầu vào: không có
- Đầu ra: không có

```
void NhapXuatTong()
{
    int x, y;
    cout<<"Nhap 2 so nguyen:";
    cin>> x >> y;
    cout<<"tong x+y la:"<< x+y;
}
```

So Sánh

```
void XuatTong(int x, int y)
{
    int s;
    s = x + y;
    cout<<"tong la:"<<s;
}
```

```
int TinhTong(int x, int y)
{
    int s;
    s = x + y;
    return s;
}
```

```
void NhapXuatTong()
{
    int x, y;
    cout<<"Nhap 2 so nguyen:";
    cin>> x >> y;
    cout<<"tong x+y la:"<< x+y;
}
```

Lưu ý: định nghĩa hàm

- KHÔNG được phép đặt định nghĩa hàm này trong một hàm khác.

```
int TinhTong(int x, int y)
{
    int s;
    s = x + y;

    int TinhHieu(int x, int y)
    {
        //…
    }

    return s;
}
```

Không được phép định nghĩa hàm tại đây

Khai báo hàm (prototype)

- Khai báo hàm là đưa ra một "mẫu hàm" (**prototype**): mô tả tên hàm, kiểu trả về và danh sách tham số.
  - **Trong khai báo prototype hàm cần lưu ý: không chứa phần thân hàm.** Nội dung của hàm sẽ được triển khai sau.
  - **Kết thúc khai báo hàm với dấu chấm phẩy ";".**
  - Thường khai báo hàm ở đầu file, phía sau phần khai báo thư viện.
- Dạng khai báo hàm:

<return_type> func_name ([ParameterList]);

Ví dụ :

```
void f1(int i, int j, float k);
double tong(double x, double y);
int giai_thua( int n) ;
void f2(int a, b, float c); //?? SAI
```

Khai báo hàm (prototype)

- Thông thường người ta thường đặt phần tiêu đề hàm/nguyên mẫu hàm (prototype) trên hàm main và phần định nghĩa hàm dưới hàm main.

```
void XuatTong(int x, int y);  // prototype

int main()
{
   …
}

void XuatTong(int x, int y)
{
   cout<<"tong la:"<<x+y;
}
```

Gọi hàm

- Chỉ với khai báo và định nghĩa hàm thì hàm đó *chưa thực thi*. ***Hàm chỉ thực*** thi khi nó ***được gọi.***
- Để *"yêu cầu" một hàm thực thi, ta "gọi" tên hàm cùng với các* tham số thực sự như một câu lệnh:

func(arg1, arg2,…); ***//Lưu ý: không có kiểu dữ liệu***

Ví dụ:

```
int c = TinhTong(x,y);
XuatTong(x,y);
int ketqua = giai_thua(n);
```

Lưu ý: Gọi hàm

1. Gọi tên của hàm đồng thời truyền các đối số (hằng, biến, biểu thức) cho các tham số **theo đúng thứ tự** đã được khai báo trong hàm.

Ví dụ:

```
int TinhHieu(int x, int y)
{
    int s;
    s = x - y;
    return s;
}
```

```
int x = 10, y = 5;
int c = TinhHieu(x,y);
int d = TinhHieu(y,x);
```

Lưu ý: Gọi hàm

2. Nếu **có phát biểu gọi hàm trước khi hàm được định nghĩa** thì **cần có một khai báo hàm (prototype) trước lời gọi hàm đó**. Nếu không chương trình sẽ báo lỗi hàm chưa được định nghĩa

Ví dụ:

```
int main()
{
    int x = 2, y = 5;
    XuatTong(x, y);  // Lỗi hàm chưa định nghĩa
}
void XuatTong(int x, int y)
{
    cout<<"tong la:"<<x+y;
}
```

Lưu ý: Gọi hàm

- Khắc phục: có 2 cách

```
// Cách 1: dùng prototype

void XuatTong(int x, int y);
int main()
{
    int x = 2, y = 5;
    XuatTong(x, y);
    return 0;
}
void XuatTong(int x, int y)
{
    cout<<"tong la:"<<x+y;
}
```

```
// Cách 2: định nghĩa hàm
trước lời gọi hàm (viết theo
thứ tự trên xuống)

void XuatTong(int x, int y)
{
    cout<<"tong la:"<<x+y;
}
int main()
{
    int x = 2, y = 5;
    XuatTong(x, y);
    return 0;
}
```

Ví dụ

- Viết hàm kiểm tra một số nguyên dương n có phải số nguyên tố hay không?

```
#include <iostream>
#include <math.h>
using namespace std;
int ktsnt(int); //prototype

int ktsnt(int k)// dinh nghia ham
{
        if(k<2) return 0;
        for(int i=2; i<=sqrt(k);i++)
        {
           if(k%i ==0)
                   return 0;
        }
        return 1;
}
```

```
int main()
{
   int n;
   cout<<"nhap vao n:";
   cin>>n;
   if(ktsnt(n)==1)// goi ham
      cout<<n<<" la so nguyen to" <<endl;
   else
      cout<<n<< " khong la so nguyen to"<<endl;
   return 0;
}
```

Ví dụ

- Viết hàm tính tổ hợp chập k của n

```
#include <iostream>
#include <math.h>
using namespace std;
long giaithua(int ); //prototype
int tohop(int , int ); //prototype
long giaithua(int k)
{
   long kq = 1;
   for(int i=1;i<=k;i++)
      kq = kq *i;
   return kq;
}
int tohop(int n, int k)
{
   return giaithua(n)/(giaithua(k)*giaithua(n-k));
}
int main()
{
   int n, k;
   cout<<"nhap vao n va k tuong ung:";
   cin>>n>>k;
   cout<<"ket qua la:" << tohop(n,k)<<endl;
   return 0;
}
```

Tham số trong chương trình con

- Chương trình con có thể không cần tham số mà chỉ có các biến riêng (biến cục bộ).
- Trường hợp **cần chuyển các giá trị cho hàm khi gọi hàm** thì cần **định nghĩa danh sách tham số của hàm**, còn gọi là các ***tham số hình thức***.
- Mỗi **giá trị thực** chuyển cho hàm khi gọi hàm được gọi là ***đối số (hay tham số thực).***
- Mỗi khi gọi hàm, có thể chuyển các đối số khác nhau.

```
void XuatTong(int x, int y);
int main()
{
    int a = 2, b = 5;
    XuatTong(a, b);  // a+b
    return 0;
}
void XuatTong(int x, int y)
{
    cout<<"tong la:"<<x+y;
}
```

đối số (hay tham số thực)

tham số hình thức

Truyền tham số

- Để một hàm thực thi, cần **gọi hàm** với tên và **chuyển các đối số** tương ứng với **danh sách tham số hình thức cả về kiểu và thứ tự**.
- Có 3 cách truyền tham số cho hàm:
  - Truyền bằng tham trị (Call by Value)
  - Truyền bằng tham chiếu (Call by Reference)
  - Truyền bằng địa chỉ (Call by Address): *học ở môn học Kỹ thuật lập trình*

Truyền tham số

- **Truyền bằng tham trị** (Call by Value)
  - Là khi **giá trị của đối số được *sao chép vào cho tham số* hình thức**. Như vậy, **các thay đổi cho tham số hình thức (trong hàm) KHÔNG làm thay đổi đối với tham số thực**.
  - Mặc định, với cách khai báo danh sách tham số với kiểu và tên, ta có cách chuyển tham trị.

Ví dụ: truyền tham trị

```
#include <iostream>
#include <math.h>
using namespace std;
void thamtri(int, int);
void thamtri(int ix, int iy)
{
      ix += 1; //cong ix them 1
      iy += 1; //cong iy them 1
}
int main()
{
     int ia = 5, ib = 5;
      thamtri(ia, ib);
      // ia = 5 ; ib = 5
      cout<<"ia="<<ia<<endl;
      cout<<"ib="<<ib<<endl;
     return 0;
}
```

các thay đổi cho tham số hình thức (trong hàm, ix và iy) không làm thay đổi đối với tham số thực (ia và ib)

Khi thực hiện xong hàm thamtri(), giá trị ia và ib **không thay đổi** do truyền tham số bằng tham trị

Truyền tham số

❑ **Toán tử địa chỉ và tham chiếu**

- Toán tử **&** (**address-of operator**) được sử dụng để lấy địa chỉ của một biến trong bộ nhớ. '**&**' đặt trước tên biến.
- Tham chiếu (**Reference – ký hiệu &**): Mục đích của tham chiếu trong C++ là tạo ra **một biến khác** có **cùng kiểu dữ liệu** nhưng **sử dụng chung vùng nhớ (địa chỉ) với biến được tham chiếu đến.**
- Cú pháp: <data_type> & <tên_tham_chiếu> = <biến_được_tham_chiếu>;

Ví dụ:

```
int x = 5;
cout<<x;  // in ra giá trị của biến x là 5
cout<<&x; // in ra địa chỉ của biến trong vùng nhớ chẳng hạn 0x7ffe9173661c
int &r = x;  // biến r tham chiếu tới giá trị biến x, lưu ý không phải là địa chỉ của x
```

Lưu ý: address-of operator và **reference** dùng chung 1 ký hiệu **&**

Như vậy, mọi hành vi thay đổi giá trị của ***r*** đều tác động trực tiếp đến **x**.
Lưu ý: Biến tham chiếu sẽ có địa chỉ cố định sau khi khởi tạo. Chúng ta không thể tham chiếu lại lần nữa.

Truyền tham số

- **Tham chiếu** thường được sử dụng chính:
  - Đối với tham số của hàm.
  - Trong kiểu trả về của hàm.
  - Cho các phép toán *"nạp chồng"*.
- Lưu ý :
  - Không tham chiếu đến biến ***khác kiểu.***

```
double x;
int &n = x; //??? ERROR
```

  - Không tham chiếu đến hằng.

```
const int a = 5;
int &r = a; //??? ERROR
int &t = 7; //??? ERROR
```

```
#include <iostream>
using namespace std;
int main()
{
   int n = 3, m =7;
   int &r = n;
   cout<<"r =" <<r<<"\t n =" <<n<<endl;
   r = m ;
   cout<<"r =" <<r<<"\t n =" <<n<<endl;
   n = n+7;
   cout<<"r =" <<r<<"\t n =" <<n<<endl;
   cout <<"m="<<m;
   return 0;
}
```

Truyền tham số

- **Truyền bằng tham chiếu (Call by Reference)**
  - Khi muốn **tham số hình thức và tham số thực cùng địa chỉ** (bản chất là cùng ô nhớ nhưng khác tên), ta dùng cách **chuyển tham chiếu cho hàm.**
  - Khai báo tham số của hàm với kí tự '&' ngay trước tên (giữa kiểu dữ liệu và tên).
  - Như vậy, khi **dùng tham chiếu mọi thay đổi đối với tham số hình thức cũng làm thay đổi tham số thực.**
  - Có thể **dùng chuyển tham chiếu để *trả về giá trị cho* nơi gọi hàm**.

Ví dụ: Truyền tham chiếu

```
#include <iostream>
using namespace std;
void thamchieu(int &, int &);
void thamchieu(int &ix, int &iy)
{
       ix += 1; //cong ix them 1
       iy += 1; //cong iy them 1
}
int main()
{
      int ia = 5, ib = 5;
       thamchieu(ia, ib);
       // ia = 6 ; ib = 6
       // ia va ib  thay doi theo ix va iy
       cout<<"ia="<<ia<<endl;
       cout<<"ib="<<ib<<endl;
     return 0;
}
```

tham số hình thức và tham số thực cùng địa chỉ
mọi thay đổi đối với tham số hình thức cũng làm thay đổi tham số thực.

Khi thực hiện xong hàm thamchieu(), giá trị ia và ib **thay đổi theo giá trị ix và iy**

Câu hỏi

- Cho biết kết quả in ra màn hình của đoạn chương trình sau đây

```
#include <iostream>
using namespace std;
void hoanvi_1(int , int );
void hoanvi_2(int &, int &);

void hoanvi_1(int x, int y )
{
   int temp = x;
   x = y;
   y = temp;
}
void hoanvi_2(int &x, int &y)
{
      int temp = x;
   x = y;
   y = temp;
}
```

```
int main()
{
   int a = 2, b = 7;
   hoanvi_1(a,b);
   cout<<"a="<<a<<"\t b="<<b<<endl;
   hoanvi_2(a,b);
   cout<<"a="<<a<<"\t b="<<b<<endl;
   return 0;
}
```

Lưu ý: truyền tham chiếu

1. Trong một hàm, các tham số có thể truyền theo nhiều cách

```
void HonHop(int x, int &y)
{
    …
    x++;
    y++;
}
```

Lưu ý: truyền tham chiếu

2. Dùng chuyển tham chiếu để *trả về giá trị cho* nơi gọi hàm.

```
#include <iostream>
using namespace std;
void XuatTong(int x, int y);
void TinhTong_Ref(int x, int y, int &tong);

void XuatTong(int x, int y)
{
	cout<<"tong la:"<<x+y<<endl;
}
void TinhTong_Ref(int x, int y, int &tong)
{
	tong = x+y;
}
```

```
int main()
{
	int a = 2, b = 5;
	XuatTong(a, b);  // a+b
	int ketqua = 0;
	TinhTong_Ref(a,b,ketqua);
	cout<<"tong la:" << ketqua;
	return 0;
}
```

Giá trị trả về

- Một hàm không trả về giá trị khi hàm được khai báo có kiểu là **void.**
- Ngược lại, hàm phải **trả về giá trị có kiểu cùng với kiểu trả về đã khai báo**.
- **Lệnh return nhằm dừng thực thi hàm, trở về** nơi gọi nó; và còn được dùng để trả giá trị (tính toán được) về cho nơi gọi hàm.
- Trong một hàm, có thể có nhiều lệnh return, nhưng chỉ 1 lệnh return được thực thi.
- Trong một lệnh return **chỉ có 1 giá trị** được trả về.

Ví dụ

Tính x^n

```
#include <iostream>
using namespace std;
// khai bao prototype
int power(int, int);
void power_2(int, int);
// ham tinh so mu
int power(int ix, int in)
{
	int i, ip = 1;
	for(i = 1; i <= in; i++)
		ip *= ix;
	return ip;
}
```

```
void power_2(int ix, int in)
{
	int i, ip = 1;
	for(i = 1; i <= in; i++)
		ip *= ix;
	cout<<"ket qua:" <<ip<<endl;
}
int main()
{
	int kq = power(2, 3);
	cout<<"2^3 =" <<kq<<endl;

	power_2(2,3);
	return 0;
}
```

Câu hỏi

- Trong các định nghĩa hàm sau đây, có những định nghĩa hàm không hợp lệ.

```
int f1() {} //????
void f2() { return 0;} //????
int f3() { return ; } //????
void f4() { return;} //????
int f5() { return 0;} //????
int f6(){ return 0.5;} //????
```

Ví dụ:

- Viết hàm kiểm tra một số có phải là số chính phương hay không?

(Trong một hàm, có thể có nhiều lệnh return, nhưng chỉ 1 lệnh return được thực thi)

```
#include <iostream>
#include <math.h>
using namespace std;
int ktscp(int );
int ktscp(int k)
{
      if (sqrt(k) == int(sqrt(k)))
            return 1;
      else
            return 0;
}
int main()
{
   int a;
   cin>>a;
   if(ktscp(a)==1)
      cout<<a<<" la so chinh phuong";
   else
      cout<<a<<" khong la so chinh phuong";
      return 0;
}
```

Phạm vi của biến

- Có 3 nơi cơ bản mà biến được khai báo:
  - trong một hàm,
  - trong định nghĩa danh sách tham số của hàm,
  - ngoài tất cả các hàm.
- Tương ứng với vị trí xuất hiện của biến, có:
  - Biến địa phương/cục bộ (local variable)
  - Tham số hình thức (argument parameter)
  - Biến toàn cục (global variable)

Biến địa phương

- Các biến (hằng) được khai báo ***trong*** *một hàm* được gọi là các *biến địa phương*.
- Biến có thể được khai báo bất kì đâu trong hàm, chỉ các lệnh trong *cùng khối mới có thể truy* xuất.
- Biến địa phương chỉ tồn tại (thời gian sống) trong khi *khối lệnh có chứa khai báo biến đó thực thi*.
- Khối lệnh hoặc hàm khác không thể truy xuất chúng.

Tham số hình thức

- Dùng tham số hình thức để chuyển các giá trị cho hàm.
- Các tham số hình thức được dùng *như biến địa phương*. *Nghĩa là:*
  - biến chỉ được sinh ra khi hàm được gọi thực thi và bị hủy khi hàm thực thi xong.
  - Chỉ được truy xuất bởi các lệnh trong hàm đó.

Biến toàn cục

- Để tạo biến (hằng) toàn cục, khai báo biến (hằng) **ngoài** tất cả các hàm.
- Biến toàn cục có thể được truy xuất bởi các lệnh ở bất kì đâu trong chương trình kể từ sau khi nó được định nghĩa (khai báo).
- Thời gian sống của biến toàn cục là suốt quá trình chương trình thực thi.

Câu hỏi:
Hãy cho biết biến nào là biến toàn cục, biến cục bộ và tham số hình thức ?

```
int a;
    int Ham1(int x, int y)
    {
        int a1;
    }

    int Ham2()
    {
        int a2;
        {
            int a21;
        }
    }

    int main()
    {
        int a3;
    }
```

Trường hợp trùng tên biến

- Không thể định nghĩa hai biến trùng tên trong cùng khối.
- Nếu có biến địa phương trong hàm trùng tên với biến toàn cục, thì trong hàm đó, **mặc định sẽ truy xuất đến biến địa phương**.
- Để truy xuất đến biến toàn cục, ta dùng **phép toán phân định phạm vi**, là dấu hai chấm kép [**::**] ngay trước tên biến.

Ví dụ:

```
#include <iostream>
using namespace std;
void local_func();
int global_var = 10; // Biến toàn cục
int main()
{
        cout<<"global var = " << global_var <<endl; // 10
        local_func();
        // ra khoi ham local_func()
        cout<<"global var = " << global_var<<endl; // 20
        return 0;
}
void local_func()
{
      cout<<"global var in locAl function = " << global_var <<endl; // 10
        int global_var = 5; // biến cục bộ
        cout<<"local var = " <<global_var <<endl; //5
        ::global_var = 20;
        cout<<"local var = " <<global_var <<endl; //5
        cout<<"global var in locAl function = " << ::global_var <<endl; //20
}
```

Đệ quy

❖ Khái niệm

- Một chương trình con có thể gọi một chương trình con khác.
- Nếu gọi chính nó thì được gọi là sự đệ quy.
- Số lần gọi này phải có giới hạn (điểm dừng)

Ví dụ

Tính S(n) = n! = 1*2*…*(n-1)*n

Ta thấy S(n) = S(n-1)*n

Vậy thay vì tính S(n) ta sẽ đi tính S(n-1)

Tương tự tính S(n-2), …, S(2), S(1), S(0) = 1

*Học sâu hơn ở môn kỹ thuật lập trình

Ví dụ: Đệ quy

```
int GiaiThua(int n)
{
   if (n == 0)
      return 1;
   else
      return GiaiThua(n - 1) * n;
}
int GiaiThua(int n)
{
   if (n > 0)
      return GiaiThua(n - 1) * n;
   else
      return 1;
}
```

Macro vs. Function

- Marco là 1 tên bất kì (do lập trình viên đặt tên) trỏ tới 1 khối lệnh thực hiện một chức năng nào đó
- Macro được dùng để chỉ những định nghĩa được viết ở phần tiền xử lý (pre-processor), thay vì đặt nó vào trong phần thực thi của file nguồn. Trong quá trình tiền xử lí, các macro được sử dụng trong chương trình được thay thế bởi các khối câu lệnh tương ứng
- Cú pháp: **#define <ten_macro> ([danh _tham _so (neu_co)]) (cau_lenh)**
- Macro thường đặt ở đầu file, chung phần khai báo thư viện. Không có dấu chấm phẩy ở cuối macro.

Ví dụ:

```
#define MAX(A, B) ((A) > (B) ? (A) : (B))     // macro có tham số
#define PRINT_HELLO_WORLD printf("Hello world")    // macro không tham số
```

Macro

Ví dụ: Macro có tham số

a) Tìm số lớn nhất của hai số

```
#define MAX(A, B) ((A) > (B) ? (A) : (B))
```

b) Tìm số nhỏ nhất của hai số

```
#define MIN(A, B) ((A) < (B) ? (A) : (B))
```

b) Tính tổng hai số

```
#define TONG_HAI_SO(x, y)  (x) +(y)
```

Ví dụ: Macro

- Ví dụ: Macro không tham số

#define PRINT_HELLO_WORLD printf("Hello world")

```
#include <stdio.h>
#define PRINT_HELLO_WORLD printf("Hello world")
int main()
{
   PRINT_HELLO_WORLD;
   return 0 ;
}
```

Ví dụ: Macro

```
#include <stdio.h>
#define TONG_HAI_SO(x, y) (x) + (y)
int main()
{
	int a = 10;
	int b = 20;
	int tong = 0;
	 tong = TONG_HAI_SO(a,b);
	printf("\n Tong cua %d va %d la %d", a, b, tong);
	float c = 10.5;
	float d = 15.3;
	float sum = 0;
	sum = TONG_HAI_SO(c,d);
	printf("\n Tong cua %.3f va %.3f la %.3f", c, d, sum);
	return 0 ;
}
```

Macro

- **Chú ý: khi sử dụng macro để tránh side effect**

```
#include <stdio.h>
#define SQUARE(X) (X*X)
int main()
{
   printf("SQUARE(%d) = %d", 3+5, SQUARE(3+5));
   return 0;
}
```

Giải thích:
Kết quả mong muốn sẽ là 8*8 = 64. Nhưng thực tế, kết quả như sau: **3+5*3+5 = 3 + 15 + 5 = 23**.
Khi bạn định nghĩa macro, phải chú ý dấu ngoặc. Macro SQUARE được update lại như sau:

#define SQUARE(X) ((X)*(X))

Ví dụ: Macro vs. Function

Ví dụ: tìm số lớn nhất giữa 2 số a và b

Macro

```
#include <stdio.h>
#define MAX(A, B) ((A) > (B) ? (A) : (B))

int main()
{
   int a = 5, b = 7;
   float c = 5.6, d = 4.5;
   printf("\nMAX= %d",  MAX(a, b));
   return 0 ;
}
```

Function

```
#include <stdio.h>
int timMax(int, int); // prototype
int main()
{
   int a = 5, b = 7;
   printf("\nMAX= %d", timMax(a, b));
   return 0 ;
}
int timMax(int x, int y)
{
   return  ((x) > (y) ? (x) : (y));
}
```

Gọi hàm

Định nghĩa hàm

Macro vs. Function

- **Ưu nhược điểm của Macro và Hàm**

Macro	Function
Việc định nghĩa macro khó hơn định nghĩa hàm. Nếu không chú ý, chúng ta dễ bị **side effect** **#define SQUARE(X) (X*X)** **SQUARE(3+5) -> 3+5*3+5 = 3 + 15 + 5 = 23**	Việc định nghĩa đơn giản hơn
Không thể debug tìm lỗi của macro trong thời gian thực thi.	Debug đơn giản, dễ bắt lỗi
Macro không cần quan tâm kiểu dữ liệu của tham số và kiểu trả về. Như ví dụ trên, chúng ta có thể truyền kiểu int, float.	Phải chỉ rõ kiểu dữ liệu của tham số và giá trị trả về
Thời gian xử lí ngắn hơn thời gian gọi hàm	Chương trình mất time dịch từ vùng nhớ hàm được lưu trữ sang vùng nhớ goi hàm.
Giả sử macro được gọi 20 lần trong chương trình, 20 dòng code sẽ được chèn vào chương trình trong quá trình tiền xử lí. Điều này làm cho kích thước của chương trình (.EXE, .DLL, .LIB,…) phình to ra.	Giả sử 1 hàm được gọi 20 lần, sẽ chỉ có 1 bản copy của hàm trong chương trình. Kích thước chương trình nhỏ hơn sử dụng macro.

Các ví dụ minh họa

- Viết chương trình tính tổng:

  $S = 1 + 1/2 + 1/3 + \ldots + 1/n$

```
#include <iostream>
using namespace std;
float tinhtong (int);
float tinhtong(int n)
{
   float tong= 0.0;
       for (int i =1; i <=n; i++)
       {
              tong = tong + (1.0/i);
       }
   return tong;
}
int main()
{
       int n;
       cin >> n;
       cout <<"ket qua:" << tinhtong(n) <<endl;
       return 0;
}
```

Các ví dụ

THCSLT3-04

Cho S sẽ được tính theo công thức S=1+3+5+...+(2n+1) với n≥0 và số nguyên M. Hãy tìm giá trị n nhỏ nhất sao cho S > M với M được cho trước. Ví dụ: M = 20 thì n tìm được là 4.

```
#include <iostream>
using namespace std;
int tim_n(int);
int tim_n(int M)
{
	int tong = 0;
	 int n = 0;
	 while(1) // vong lap vo tan, dung break thoat ra
	 {
		tong = tong + (2*n+1);
		if (tong > M) // dieu kien stop
			break;
		else
			n++;
	 }
	 return n;
}
```

```
int main()
{
	int M;
	cin >> M;
	cout <<"ket qua:" << tim_n(M)<<endl;
	return 0;
}
```

Các ví dụ minh họa

- Viết chương trình kiểm tra 1 số có phải số đối xứng hay không?

Ví dụ

6 là số đối xứng

12 không đối xứng

121 là đối xứng

1220 không đối xứng

2332 là đối xứng

```
#include <iostream>
using namespace std;
int ktdoixung(int);
int ktdoixung(int n)
{    int temp = n;
     int sodaonguoc = 0;
    while(n!=0)
    {
        sodaonguoc = sodaonguoc*10 + n%10;
        n = n/10;
     }
     if(sodaonguoc==temp)
          return 1; // dx
     return 0; // khong dx
}
int main()
{      int M;
       cin >> M;
       if(ktdoixung(M))
          cout<<M<<" la so dx"<<endl;
       else
          cout<<M<<" Khong la so dx"<<endl;
       return 0;
}
```

Bài Tập

❖ **Yêu cầu viết chương trình sử dụng hàm.**

1. Viết chương trình in ra tất cả các số nguyên tố nhỏ hơn số nguyên dương M cho trước (M nhập từ bàn phím).

2. Viết hàm kiểm tra một số nguyên dương có phải là số chính phương hay không. In tất cả các số chính phương trong khoảng (A,B). (với A, B nhập từ bàn phím)

3. Viết chương trình nhập số nguyên dương N, kiểm tra xem N có phải là số hạnh phúc hay không. (Số hạnh phúc là số có 2K chữ số và tổng K chữ số đầu tiên bằng tổng K chữ số cuối cùng)

Bài Tập

4. a) Viết hàm tìm ước số chung lớn nhất và bội số chung nhỏ nhất của 2 số nguyên dương

b) Viết chương trình con rút gọn một phân số.

c) Viết chương trình nhập 2 phân số **a/b** và **c/d** (nhập các số nguyên a,b,c,d).

Tính và in ra phân số tổng ở dạng phân số tối giản.

5. Viết hàm đổi một số hệ 10 sang hệ nhị phân.

6. Viết hàm phân tích số tự nhiên n thành tích các số nguyên tố.

Ví dụ : 90 = 2*3*3*5

Bài Tập

7. Liệt kê tất cả các số tự nhiên *k* thỏa mãn đồng thời các điều kiện sau:

- *k* là số có 5 chữ số;
- *k* là số nguyên tố;
- *k* là số đối xứng (*k* là số đối xứng nếu đọc xuôi hay đọc ngược các chữ số của *k* ta đều nhận được một số như nhau. Ví dụ số: 10001 là số đối xứng).

8. Viết hàm kiểm tra một số có phải là số hoàn chỉnh hay không.

Số hoàn chỉnh là số có giá trị bằng tổng các ước số của nó (không tính ước là chính số đó).

Ví dụ: 28=1+2+4+7+14 là số hoàn chỉnh

6 = 1+2+3

9. Viết hàm kiểm tra một số có phải là số Mersenne hay không. Số Mersenne là số biểu diễn được dưới dạng $k = 2^n-1$. Ví dụ: 7, 31 là các số Mersenne